Una representación generalizada en algoritmos evolutivos para una amplia variedad de problemas de scheduling

Guillermo Daniel Ordóñez
gordonez@unsl.edu.ar

Mario Guillermo Leguizamón
legui@unsl.edu.ar

LIDIC - Universidad Nacional de San Luis,
Ejército de los Andes 950,
(5700) San Luis, Argentina

VI Workshop de Agentes y Sistemas Inteligentes (WASI)

Resumen

El problema general de *scheduling* representa un gran desafío computacional dado que es un problema inherentemente difícil. Para su resolución, existe una gran variedad de enfoques cuya efectividad depende del tipo, tamaño y otras características del problema de *scheduling*. Tales enfoques incluyen métodos tradicionales de investigación operativa, búsqueda local y sus diferentes versiones (por ejemplo, simulated annealing y tabu search) y diversas metaheurísticas bio-inspiradas tales como computación evolutiva y ant colony optimization, entre otras. La propuesta aquí planteada se centra en la descripción de una representación válida para una gran variedad de clases de problemas de *scheduling*, la cual es potencialmente apta para ser utilizada en el diseño de algunas metaheurísticas, en particular, algoritmos evolutivos.

keywords: problema de *scheduling*, metaheurísticas, representación, alcanzabilidad de clases de *scheduling*.

1. Introducción

El problema de *scheduling* es de gran importancia práctica, sin embargo gran parte de los problemas de *scheduling* son del tipo $\mathcal{NP}$-Completos. En general, un problema de *scheduling* se define por medio de un conjunto T de tareas, un conjunto M de máquinas, y un grupo de restricciones que define el problema. Algunos ejemplos de restricciones son: cada tarea tiene asociado un conjunto de tareas predecesoras, no todas las tareas pueden ejecutarse en cualquier máquina, el tiempo que una tarea requiere para ser ejecutada podría variar de una máquina a otra, etc. Un *schedule* (una solución a la instancia del problema de *scheduling*) es factible si no existe superposición de intervalos de tiempo, es decir, dos tareas no son procesadas en una misma máquina al mismo tiempo, y se cumplen las restricciones adicionales especificadas para la instancia del problema que se esté resolviendo.

Dependiendo de las restricciones (principalmente de las características del conjunto de tareas predecesoras), existen varias clases de problemas de *scheduling*. La mayoría de estas clases describen problemas del tipo $\mathcal{NP}$-Completos [7] (por ejemplo, *Job Shop Scheduling* y *Open Shop Scheduling*). Por esto, para gran parte de estas clases de problemas, han sido aplicados diferentes enfoques que incluyen técnicas tradicionales de investigación operativa, búsqueda local y sus diferentes versiones (simulated annealing, tabu search)[1] y más recientemente diversas técnicas metaheurísticas que entre

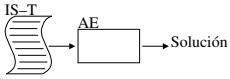

Figura 1: IS-T representa la instancia en un lenguaje de bajo nivel

otras, incluyen a los algoritmos evolutivos [2, 3, 9, 10]. La característica común de estas herramientas es que en general están diseñadas para una clase particular de problemas de *scheduling*. El objetivo de nuestra propuesta es presentar una representación adecuada para ser utilizada con diversas metaheurísticas para resolver una amplia variedad de problemas de *scheduling* de acuerdo a los modelos propuestos por Pinedo [14] y combinaciones de éstos.

En función de la propuesta cabría preguntarse: ¿cuál es la necesidad de crear una nueva herramienta para resolver problemas de *scheduling*?, Como ya se mencionó, las herramientas existentes son para problemas de *scheduling* específicos. Por lo tanto, la primer ventaja que brindará nuestra herramienta es la de su aplicabilidad a una importante familia de problemas de *scheduling*. En síntesis, nuestra herramienta apunta a ser lo suficientemente robusta y efectiva para ser aplicada a diversos tipos de problemas de *scheduling*. Debido a que gran parte de los problemas de *scheduling* pertenecen a la familia de problemas $\mathcal{NP}$-Completos y donde una solución cercana a la óptima es aceptable, los algoritmos evolutivos (AEs) conforman uno de los posibles enfoques apropiados para ser usados en este contexto.

Los AEs son algoritmos de búsqueda inspirados en la genética de las poblaciones naturales. Ellos mantienen una población de individuos que representan las soluciones candidatas. Dicha población evoluciona en el tiempo a través de la competencia entre los individuos y una variación controlada de los mismos. La aplicación de AEs incluye un amplio rango de problemas de optimización y aprendizaje de máquina en variados dominios. Los AEs han sido aplicados con diferente grado de éxito a problemas de *scheduling* de distintos tipos. Davis [6] fue el primero en proponer el uso de AEs para resolver JSS [5]. Al mismo tiempo, diferentes versiones de AEs aplicados a un problema de ordenamiento, tal como el Problema del Viajante de Comercio, dieron lugar a representaciones más avanzadas, muchas de ellas adecuadas para ser usadas en muchos problemas de *scheduling*. Un ejemplo de estas representaciones es la permutación de tareas la cual tiene asociado un conjunto considerable de operadores de entrecruzamiento a ser aplicados: PMX, OX, CX, etc. [8, 10, 13, 12].

2. Funcionamiento general de la herramienta

La herramienta está basada en un algoritmo evolutivo, el cual tiene como entrada una instancia del problema de *scheduling* descripta en un formato de bajo nivel al que se denominó IS-T, y cuya salida es un archivo que incluye la mejor solución encontrada más un conjunto de datos estadísticos útiles al momento de evaluar la performance del AE (ver figura1).

Un archivo IS-T está formado por un conjunto de tablas, lo que torna difícil su especificación y lectura, sobre todo cuando las instancias son de gran tamaño, por consiguiente, cualquier error puede llevar a alguna inconsistencia en la entrada. A fin de facilitar la tarea de especificación de las instancias, se creó un segundo lenguaje bastante intuitivo, y junto con éste se implementó un compilador que traduce las instancias de este tipo de archivos (al que a partir de ahora denominaremos IS-Sch) a un archivo del tipo IS-T (quedando la herramienta como se muestra en la figura 2). De esta forma, para obtener una solución de un problema, un usuario simplemente debería especificar el problema en el lenguaje IS-Sch e invocar la herramienta (ver figura 3). La conversión de IS-T a IS-Sch, así como la especificación completa de éstos queda fuera del alcance de este artículo. Aquí sólo se mostrarán algunas características del lenguaje IS-Sch. El artículo se centra en la codificación que

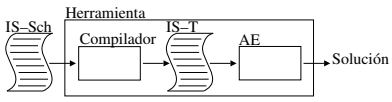

Figura 2: Incorporación de IS-Sch

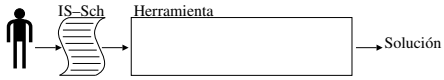

Figura 3: Herramienta desde una perspectiva global

emplea el AE para representar soluciones. Dado que para la especificación de la representación es irrelevante, la notación que utilicemos para expresar el problema de aquí en mas será a través del lenguaje IS-Sch.

3. IS-Sch: lenguaje de especificación de problemas de *scheduling*

Una instancia especificada en el lenguaje IS-Sch consta de 5 secciones. La primera es **nombre**, simplemente es utilizada para dar el nombre de la instancia, las dos siguientes **grupos** e **ini** son optativas y por simplicidad no serán incluidas en los ejemplos. La cuarta, **tareas**, contiene las tareas y las restricciones asociadas a éstas y por último, **objetivo**, el cual contiene el objetivo a minimizar. Para nuestros ejemplos utilizaremos como objetivo el *makespan*, que es simplemente el tiempo en que termina de ejecutarse la tarea que más demora.

Dentro de **tareas**, cada tarea es especificada de la siguiente forma[1]:

[⟨nombre⟩] **[** ({ ⟨máquina⟩ })
, ⟨tiempo⟩
[**,** [⟨rj⟩]]
[**,** { ⟨predecesor⟩ }]]
]

Donde *nombre* es optativo pero necesario para hacer referencia a esta tarea. Luego sigue la lista de máquinas en las que puede ejecutarse la tarea, el tiempo que insume la tarea y un campo optativo, r_j que es el mínimo tiempo de inicio de la tarea. Por último, una lista de tareas que deben terminar antes que ésta pueda iniciarse. Al crear la lista de máquinas, para hacer referencia a una máquina simplemente utilizamos su nombre, excepto cuando las distintas máquinas tengan diferentes velocidades, en cuyo caso las máquinas con diferentes velocidades deberán expresarse como un par de la forma: ({ ⟨nombre⟩**,**⟨tiempo⟩ }).

Dentro de la sección **tareas**, las tareas están separadas por . o |, indicando que las mismas deben ser realizadas en forma secuencial o paralela respectivamente. Estos operadores son binarios, y su precedencia es de izquierda a derecha en el orden en que fueron nombrados. Además de estos dos operadores, se pueden utilizar paréntesis y dos comandos especiales, **or** y **free**. El **or**, es útil cuando se quiere expresar que se debe realizar una cosa o la otra, es decir, si aparece **or(** tareas_1, tareas_2, tareas_3**)**, sólo uno de los tres grupos de tareas puede ejecutarse. Esto es útil, por ejemplo, cuando existen varias alternativas para fabricar un mismo producto. El comando **free** es utilizado en problemas del tipo *Open Shop*, para el cual varias tareas que deben ser ejecutadas, pero sus ejecuciones no pueden

[1]La siguiente expresión está incompleta ya que por simplicidad no se incluirán todas las características del lenguaje IS-Sch

entero X_MAQUINAS[*#tareas*]
permutación X_SFREE[*#frees*]
entero X_SOR[*#ors*]
entero *X_PRIORIDAD[*#máquinas*]
entero X_RETARDO[*#tareas*]

Figura 4: Estructura del cromosoma.

superponerse en el tiempo. De esta forma, si aparece **free(** tareas_1, tareas_2, tareas_3**)** todas las tareas se ejecutarán. Sin embargo, si se comenzaron a ejecutar algunas tareas del grupo tareas_2, ninguna tarea de los grupos tareas_1 y tareas_3 podrán comenzar hasta que todas las tareas del grupo tareas_2 finalicen. Además, todas esta características puedan ser utilizadas en forma anidada.

4. Representación de una solución

Cada uno de los individuos de la población tendrán asociado un cromosoma con las estructuras que se muestran en la figura 4.

Cada cromosoma es decodificado en un único *schedule* válido. Este proceso de decodificación consiste en seleccionar un elemento del conjunto de *schedules* válidos. Dicho proceso se realiza en dos etapas. La primera utiliza la información de las tres primeras estructuras del cromosoma para reducir el conjunto de soluciones, mientras que la segunda, selecciona un elemento de este subconjunto.

4.1. Etapa I

Como se mencionó previamente, esta etapa no selecciona un elemento, sino que restringe el conjunto del cual la etapa II deberá seleccionar un elemento. Esta etapa consta de tres pasos que serán explicados a través del siguiente ejemplo:

Supongamos el siguiente problema de *scheduling*:

Nombre: *problema1*
Tareas: **or(** *t1***[(***m0***),***4.0***],** *t2***[(***m1***),***1.3***],** *t3***[(***m2***),***1.1***].** *t4***[(***m8***),***1.0***])|**
*t5***[(***m0 m3***),***4.0***].** *t6***[(***m1 (m2 2) m4***),***1.3***].** *t7***[(***m8***),***1.0***]|**
free (or(*t8***[(***m2***),***3.0***],** *t9***[(***m0***),***2.0***]).** *t10***[(***m2***),***3.1***],**
*t11***[(***m3***),***3.3***].** *t12***[(***m4***),***3.0***]) ;**
Objetivo:*Makespan***.**

Aquí #tareas es 12, #free es 1 y #or es 2. Luego, para un individuo en particular, las tres primeras estructuras podrían tener los siguientes valores:

X_MAQUINAS = { 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1 }

X_SFREE = { { 2, 1} }

X_SOR = { 1, 2 }

Paso I: Observe que la tarea *t5* puede ejecutarse en m0 o m3, mientras que *t6* puede ejecutarse en m1, m2 o m4. Por lo tanto, el valor de X_MAQUINAS[5] $\in [1, 2]$. Dado que en nuestro ejemplo X_MAQUINAS[5] = 1, descartamos todos los *schedules* válidos que hayan realizado esta tarea en la máquina m3. De esta misma forma X_MAQUINAS[6] $\in [1, 3]$. Como en el ejemplo X_MAQUINAS[6] = 2, debemos descartar todas las soluciones que ejecuten la tarea t6 en la máquina m1 o m4. El resto de las tareas sólo tiene una alternativa, por lo que el único valor que puede X_MAQUINAS para las componentes asociadas a dichas tareas es 1.

Después de tomar esta serie de decisiones, el conjunto de *schedules* válidos se reduce y es equivalente al conjunto de *schedules* válidos del siguiente problema:

Nombre: *problema1a*
Tareas: **or(** *t1*[(*m0*),*4.0*], *t2*[(*m1*),*1.3*], *t3*[(*m2*),*1.1*]. *t4*[(*m8*),*1.0*])|
t5[(*m0*),*4.0*]. *t6*[(*m2*),*2.6*]. *t7*[(*m8*),*1.0*]|
free (or(*t8*[(*m2*),*3.0*], *t9*[(*m0*),*2.0*]). *t10*[(*m2*),*3.1*],
t11[(*m3*),*3.3*]. *t12*[(*m4*),*3.0*]) ;
Objetivo:*Makespan***.**

Paso II: Este paso consiste en eliminar la apariciones de la parabra clave **free**, lo primero a tener en cuenta es que en este punto, el conjunto de *schedules* válidos ha sido reducido. Para resaltar este hecho trabajaremos con el problema *problema1a* en vez del original, ya que éste representa un conjunto equivalente al que estamos haciendo referencia.

En este problema sólo hay una aparición de la palabra free, y esta tiene dos opciones ($\{1,2\}$ o $\{2,1\}$), como en el ejemplo X_SFREE[1] = $\{2,1\}$, debemos descartar del conjunto de soluciones válidas, las soluciones que realizan alguna tarea de la primer opción del free, antes que las tareas de la segunda opción.

Al igual que en el paso anterior, después de tomar esta serie de decisiones, el conjunto de *schedules* válidos se redujo, y quedo equivalente al conjunto de *schedules* válidos del siguiente problema:

Nombre: *problema1b*
Tareas: **or(** *t1*[(*m0*),*4.0*], *t2*[(*m1*),*1.3*], *t3*[(*m2*),*1.1*]. *t4*[(*m8*),*1.0*])|
t5[(*m0*),*4.0*]. *t6*[(*m2*),*2.6*]. *t7*[(*m8*),*1.0*]|
t11[(*m3*),*3.3*]. *t12*[(*m4*),*3.0*].
or(*t8*[(*m2*),*3.0*], *t9*[(*m0*),*2.0*]). *t10*[(*m2*),*3.1*] ;
Objetivo:*Makespan***.**

Paso III: Este paso consiste en eliminar las apariciones de la parabra clave **or**, nuevamente el conjunto de *schedules* válidos ha sido reducido. Para resaltar este hecho trabajaremos con el problema *problema1b*, ya que éste representa un conjunto equivalente al que estamos haciendo referencia.

En este problema hay dos apariciones de la parabra clave **or**, en la primera hay tres opciones, mientras que en la segunda sólo hay dos. Después de descartar las soluciones que no cumplan con los criterios de X_SOR, el conjunto de *schedules* válidos es equivalente al del siguiente problema:

Nombre: *problema1c*
Tareas: *t1*[(*m0*),*4.0*]|
t5[(*m0*),*4.0*]. *t6*[(*m2*),*2.6*]. *t7*[(*m8*),*1.0*]|
t11[(*m3*),*3.3*]. *t12*[(*m4*),*3.0*]. *t9*[(*m0*),*2.0*]. *t10*[(*m2*),*3.1*] ;
Objetivo:*Makespan***.**

4.2. Etapa II

Como resultado de la etapa anterior se llega a la especificación de un problema cuyo conjunto de *schedules* válidos está incluido en el problema original. Por lo tanto, si seleccionamos un *schedule* válido para este problema, estamos seleccionando un *schedule* que es válido para el problema original. Por simplicidad, en esta sección cambiaremos el problema que estamos tratando y asumiremos que el problema resultante de la etapa anterior es el siguiente:

Tareas: **:***a0*[(*m1*),*1.0*] . **:***a1*[(*m0*),*1.0*] |
:*b0*[(*m1*),*1.0*] . **:***b1*[(*m0*),*1.0*] |
:*c0*[(*m0*),*2.0*] . **:***c1*[(*m1*),*2.0*] ;
Objetivo:*Makespan***.**

```
// Inicializa el tiempo en 0
T := 0
// Inicializa las Máquinas
for i := 0 to #máquinas − 1 do
 M[i] = {}
while not fin do
 begin
  Listas = "Tareas listas para ser realizadas"
  for i := 0 to #máquinas − 1 do
   begin
    // Si la máquina i esta libre
    if Max(t_fin/(tarea, [t_inicio, t_fin)) ∈ M[i]) <= T then
    begin
     // Tareas Listas para ejecutarse en la máquina i
     Tareas = {tarea/(tarea, i) ∈ Listas}
     // Asocia a las tareas una prioridad
     Tareas = Tareas ⋈ Prioridad[i]
     // Selecciona la tarea de mayor prioridad
     tarea_seleccionada/
      (tarea_seleccionada, Max(p/(tarea, p) ∈ Tareas)) ∈ Tareas
     // Asigna la máquina
     M[i] = M[i]∪
      {(tarea_seleccionada, [T, T + Tiempo[tarea_seleccionada]))}
    end
   end
 end
```

Figura 5: Algoritmo de schedule builder.

4.2.1. Utilizando sólo X_PRIORIDAD

En este problema podemos ver que hay dos máquinas (m0 y m1) y seis tareas (a0, a1, b0, b1, c0 y c1) para ser ejecutadas sobre cada máquina. Es por esto que X_PRIORIDAD es un arreglo de 2 (cantidad de máquinas) permutaciones, en este caso las 2 permutaciones son de tamaño 3. Para nuestro ejemplo asumamos X_PRIORIDAD = {{c0,b1,a1},{a0,c1,b0}}. [2]

Para armar el *schedule* se ejecuta el algoritmo de la figura 5. Para llevar el control de la ejecución, tendremos en cuenta el contenido de:

- t: tiempo actual de la simulación,
- m[i]: lista de tareas ya asignadas a la máquina m_i,
- listas: conjunto de pares *(t, m)* tal que la tarea *t* está lista para ser asignada a la máquina *m*

Inicialmente:

t=0
listas={(a0,m1),(b0,m1),(c0,m0)}
m[0]={}
m[1]={}

En este momento la máquina m1 está libre y hay dos tareas listas para ejecutar en la misma, {(a0,m1),(b0,m1)}. La máquina m0 también está libre y hay una tarea lista para ejecutar, {(c0,m0)}. El caso de la máquina m0 es simple, se asigna dicha máquina en el intervalo de tiempo [0,2) a la tarea c0, quedando las estructuras de la siguiente forma:

[2] En realidad X_PRIORIDAD = {{4,3,1},{0,5,2}}, en este ejemplo utilizaremos los nombre por claridad.

t=0
listas={(a0,m1),(b0,m1)}
m[0]={(c0,[0,2))}
m[1]={}

Para problemas como el de la máquina m1 donde las tareas a0 y b0 están listas para ejecutar, utilizamos las prioridad dadas por X_PRIORIDAD para decidir cuál tarea asignar, en este caso como a0 está antes que b0 en X_PRIORIDAD, asignamos la máquina a a0. De esta manera las estructuras se modifican de la siguiente forma:

t=0
listas={(b0,m1)}
m[0]={(c0,[0,2))}
m[1]={(a0,[0,1))}

En este punto están todas las máquinas ocupadas, por lo que avanzamos el tiempo hasta el mínimo tiempo de liberación de alguna máquina, en este caso 1, el tiempo de liberación de m1.

t=1
listas={(b0,m1), (a1,m0)}
m[0]={(c0,[0,2))}
m[1]={(a0,[0,1))}

En este tiempo la única máquina libre es m1. Por lo tanto, se le asigna la única tarea esperando por ésta. Después de realizar esta operación y avanzar el tiempo, las estructuras quedan de la siguiente forma:

t=2
listas={(b1,m0), (a1,m0), (c0,m1)}
m[0]={(c0,[0,2))}
m[1]={(a0,[0,1)), (b0,[1,2))}

t=3
listas={(a1,m0)}
m[0]={(c0,[0,2)), (b1,[2,3))}
m[1]={(a0,[0,1)), (b0,[1,2)), (c1,[2,4))}

t=4
listas={}
m[0]={(c0,[0,2)), (b1,[2,3)), (a1,[3,4))}
m[1]={(a0,[0,1)), (b0,[1,2)), (c1,[2,4))}

Observe que en este momento la simulación terminó dejando la solución en m[0] y m[1].

4.2.2. La estructura X_RETARDO

El algoritmo anterior limita su salida al tipo de *schedules nondelay*. Para extender este conjunto, se incluye la estructura X_RETARDO. Supongamos el siguiente ejemplo:

Tareas: **:***a0***[(***m2***),***1.0***]** . **:***a1***[(***m1***),***2.0***]** . **:***a2***[(***m0***),***1.0***]** |
:*b0***[(***m0***),***2.0***]** . **:***b1***[(***m1***),***1.0***]** . **:***b2***[(***m2***),***3.0***]** **;**
Objetivo:*Makespan***.**

Sin importar los valores de X_PRIORIDAD, el único resultado posible de la simulación sin utilizar x_RETARDO es:

// Inicializa el tiempo en 0
$T := 0$
// Inicializa las Máquinas
for $i := 0$ to $\#máquinas - 1$ do
$M[i] = \{\}$
while not fin do
begin
$Listas$ = "Tareas listas para ser realizadas"
for $i := 0$ to $\#máquinas - 1$ do
begin
// Si la máquina i esta libre
if $Max(t_{fin}/(tarea, [t_{inicio}, t_{fin})) \in M[i]) <= T$ then
begin
// Tareas Listas para ejecutarse en la máquina i
$Tareas = \{tarea/(tarea, i) \in Listas\}$
// Asocia a las tareas una prioridad
$Tareas = Tareas \bowtie Prioridad[i]$
// Selecciona la tarea de mayor prioridad
$tarea_{seleccionada}/$
$(tarea_{seleccionada}, Max(p/(tarea, p) \in Tareas)) \in Tareas$
// Verifica retardo
if $Retardo[tarea_{seleccioanda}]$--= 0 then
begin
// Asigna la máquina
$M[i] = M[i] \cup$
$\{(tarea_{seleccionada}, [T, T + Tiempo[tarea_{seleccionada}]))\}$
end
end
end
end

Figura 6: Algoritmo de Schedule builder, utilizando X_RETARDO.

t=7
listas={}
m[0]={(b0,[0,2)), (a2,[3,4))}
m[1]={(a1,[1,3)), (b1,[3,4))}
m[2]={(a0,[0,1)), (b2,[4,7))}

Si realizamos la simulación paso a paso al siguiente punto:

t=1
listas={(a1,m1)}
m[0]={(b0,[0,2))}
m[1]={}
m[2]={(a0,[0,1))}

En este punto se debería asignar a la tarea a1 a la máquina m1. Asumamos que X_RETARDO es {0,1,0,0,0,0}, es decir vale 0 para todas las tareas excepto para a1. Luego de seleccionar la tarea a asignar se verifica si el valor de retardo correspondiente a dicha tarea es 0. De no ser así, se decrementa ese valor y se deja la máquina libre aunque haya otras tareas listas para ejecutarse sobre esa máquina (ver el algoritmo de la figura 6). En este caso, el algoritmo pasaría por los siguientes estados:

t=1
listas={(a1,m1)}
m[0]={(b0,[0,2))}
m[1]={}
m[2]={(a0,[0,1))}
retardo={0,1,0,0,0,0}

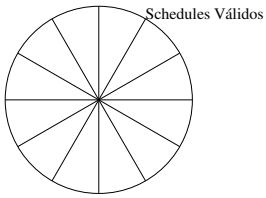

Figura 7: División del conjunto de *schedules* válidos generado por la primer parte del cromosoma

t=1
listas={}
m[0]={(b0,[0,2))}
m[1]={}
m[2]={(a0,[0,1))}
retardo={0,**0**,0,0,0,0}

t=2
listas={(a1,m1) (b1,m1)}
m[0]={(b0,[0,2))}
m[1]={}
m[2]={(a0,[0,1))}
retardo={0,0,0,0,0,0}

En este punto, las tareas a1 y b1 están listas para utilizar la máquina m1, por lo que la tarea que esté primero en X_PRIORIDAD es la que será asignada. si asumimos que b1 es la de mayor prioridad, la simulación final quedaría de la siguiente forma:

t=7
listas={}
m[0]={(b0,[0,2)), (a2,[5,6))}
m[1]={(b1,[2,3)), (a1,[3,5))}
m[2]={(a0,[0,1)), (b2,[3,6))}

Observe que a pesar de que éste no es un *schedule nondelay*, el valor de *makespan* conseguido es menor.

4.3. Alcanzabilidad de la representación

Para analizar la alcanzabilidad de la representación, es necesario en primer lugar, analizar la etapa I. Si hay dos cromosomas diferentes, después de realizar dicha etapa, los subconjuntos de *schedules* válidos generados a partir de estos cromosomas son los mismos o son disjuntos. Además, dado cualquier *schedule* válido *s*, fácilmente pueden generarse los valores para las tres primeras estructuras del cromosoma, de forma tal que al aplicar la etapa I a este cromosoma, se obtiene como resultado un conjunto que incluya a *s*. Es decir, la etapa I consigue generar una partición del espacio de búsqueda como muestra la figura 7.

Cada uno de estos subconjuntos puede contener una cantidad infinita de elementos, lo cual hace imposible encontrar una representación finita que permita representar todos los elementos de dichos subconjuntos. Utilizando el algoritmo de la figura 5, que sólo utiliza la estructura X_PRIORIDAD, el conjunto de *schedules* que pueden ser representados para este subconjunto de *schedules* válidos es equivalente la de los *schedules nondelay* de este subconjunto. Sin embargo, con el algoritmo 6, al utilizar X_RETARDO, el conjunto de *schedules* que pueden ser representados pasa a incluir al de los

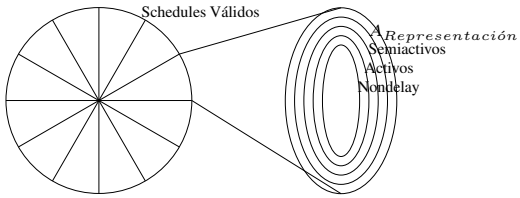

Figura 8: Alcanzabilidad de los subconjuntos creados por la primer parte del cromosoma

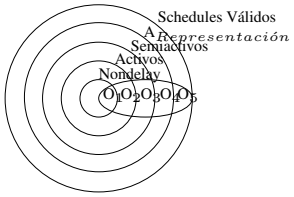

Figura 9: Óptimos

schedules semiactivos de este subconjunto ver la figura 8.

Por último, al poder representar al menos todos los *schedules* del tipo *semiactivos* de cada subconjunto, sabemos que el conjunto de *schedules* alcanzables incluye al de los *semiactivos*.

Observe en la figura 9 la relación entre el conjunto $o = o_1 \cup o_2 \cup o_3 \cup o_4 \cup o_5$ el cual representa el conjunto de todos los óptimos y $A_{Representación}$, el conjunto de *schedules* que pueden ser efectivamente representados. Puede observarse que si algún óptimo se encuentra en $o_1 \cup o_2 \cup o_3 \cup o_4$, éste podrá ser representado.

4.4. Restricciones en el espacio de búsqueda

Debido a que gran cantidad de las soluciones que podemos representar se encuentran en $A_{Representación}-$ semiactivos, una alternativa es modificar el individuo para que $A_{Representación} \equiv$ semiactivos. Con esto se consigue reducir el espacio de búsqueda de manera tal que si un óptimo se encuentra en $o_1 \cup o_2 \cup o_3$ aún podemos encontrar un óptimo facilitando en trabajo del algoritmo de búsqueda que se utilice.

Para conseguir ésto, se incorporó una función de corrección. Esta función es divide el espacio de búsqueda (Genotípico) original en clases de equivalencias y seleccionar un individuo s_i perteneciente al conjunto de los *schedules* semiactivos como representante de este conjunto (ver figura 10). Para cada individuo $g_j \in [s_i]$ se cumple que:

- si g_j representa un *schedule* semiactivo, entonces g_j representa al mismo *schedule* que s_i
- si g_j representa un *schedule* no incluido en el conjunto de *schedules* semiactivos, entonces g_j

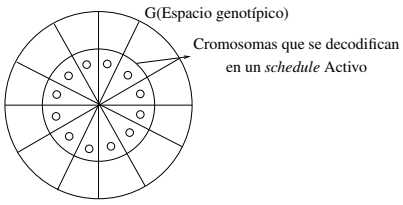

Figura 10: Clases de equivalencias sobre el espacio genotípico

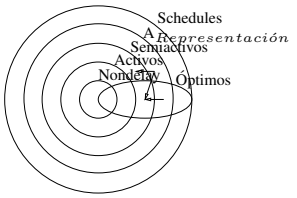

Figura 11: Movimientos en el espacio fenotípico

representa un *schedule* para el que los valores de cada una de sus variables[3] es mayor o igual que el de las de s_i, es decir que $\forall x : d(g_j)_x \geq d(s_i)_x$[4], siendo *d* una de las tres funciones en las que se descompone la función de adaptabilidad y *x* indica a cual de las variables hacemos referencia.

Con esta función se consigue:

- generar una relación uno a uno entre el espacio genotípico y el espacio fenotípico (si permitimos que la función de optimización modifique el cromosoma).
- para los individuos que no representan un *schedule* semiactivo, realizar un "movimiento" en el espacio fenotípico como se muestra en la figura 11, el cual mantiene la calidad del *schedule* generado, o lo mejora.

Es importante aclarar que la función de corrección será útil sólo para aquellos problemas en los que forzar a que una tarea comience antes, mejore o mantenga la calidad de la solución. Es decir, aquellos problemas que tengan al menos un óptimo en el conjunto $o_1 \cup o_2 \cup o_3$.

5. Discusión sobre la alcanzabilidad de la representación

En este punto se presentan cuatro casos que dependen de la función objetivo:

- Podemos asegurar a priori que $o_1 \neq \{\}$, por lo que sería conveniente utilizar el primer *schedule builder* en donde no se utiliza la parte X_RETARDO del cromosoma. En este caso, el conjunto de *schedules* que pueden ser representados utilizando este algoritmo es exactamente el conjunto de *schedules nondelay*.
- Podemos asegurar a priori que $o_1 \cup o_2 \cup o_3 \neq \{\}$. En este caso es conveniente utilizar la función de corrección propuesta (objetivos tales como makespan, o cualquier otro en donde terminar una tarea en un tiempo menor no produzca un peor resultado).
- No podemos asegurar a priori que $o_1 \cup o_2 \cup o_3 = \{\}$, pero si que exista un schedule en $o_1 \cup o_2 \cup o_3 \cup o_4$. En este caso la función de corrección *podría* no ser suficiente para representar el óptimo.
- No podemos asegurar a priori que exista algún óptimo en $o_1 \cup o_2 \cup o_3 \cup o_4$ por lo que existe la posibilidad de no poder representar un *schedule* óptimo.

[3]Las variables disponibles son los tiempos de terminación de cada tarea y máquina.

[4]Esto no es válido para toda función de adaptabilidad, más adelante se aclarará qué condiciones debe cumplir ésta para que dicha afirmación sea válida.

Primero notemos que una de las razones por la que se dividieron las estructuras que conforman el cromosoma en dos grupos es que la primer parte del mismo hace referencia al subconjunto de *schedules* válidos, mientras que la segunda, decide que solución dentro de este conjunto es utilizada. De hecho luego de terminar la etapa I cada individuo representa uno de los subconjuntos de *schedules* válidos. Esta etapa no impone ninguna restricción a la ancanzabilidad, ya que todos los *schedules* están incluidos en algún subconjunto. Sin embargo, el hecho de que cada uno de estos subconjuntos tenga una cantidad infinita de elementos hace que sea imposible encontrar una representación con individios de longitud finita que permita representat todos los *schedules* válidos. Por esta razón se tuvo que decidir que individuos iban a ser representados y se decidió no hacerlo al azar, sino en base a conocimiento del problema, asumiendo que $o_1 \cup o_2 \cup o_3 \neq \{\}$.

Aunque para gran parte de los problemas de *scheduling* encontrados en la bibliografía se puede asegurar a priori que existe al menos un óptimo en $o_1 \cup o_2 \cup o_3 \cup o_4$ o en subconjuntos de éste (por ejemplo si el objetivo a minimizar es makespan, existe al menos un óptimo en $o_1 \cup o_2$), existen problemas de *scheduling* que no poseen ningún óptimo en $o_1 \cup o_2 \cup o_3 \cup o_4$. Si éste es el caso, las representaciones propuestas no permitirán expresar los óptimos. Para este tipo de problemas, aunque la primer etapa de la decodificación permanecería igual, no sólo se deberán cambiar las dos últimas estructuras del cromosoma, también se deberá modificar el algoritmo de *schedule builder* de manera tal que se pueda asegurar a priori que con este nuevo algoritmo de *schedule builder*, se pueda alcanzar al menos un óptimo.

6. Conclusiones

En este trabajo se propuso una representación avanzada para ser usada en un AE (u otra metaheurística). Dicha representación sería aplicable a una gran variedad de problemas de *scheduling*, no sólo a los tradicionales, sino también a combinaciones de éstos. Segundo, dependiendo de las características de la función objetivo, son los componentes que conviene utilizar en la segunda parte del cromosoma, como ya dijimos, dependiendo del problema específico mucha veces podemos asegurar que $\cup_i^n o_i \neq \{\}$ para un *n* menor que 4. Si se considera a n como un parámetro adicional del algoritmo, se podría guiar la búsqueda de una manera eficiente seleccionando un *n* lo suficientemente grande como para poder asegurar que algún óptimo podrá ser representado, aunque un n grande incrementará considerablemente el espacio de búsqueda.

Referencias

[1] Aarts, E. & Lenstra, J.K. - Editores, (1997) - Local search in Combinatorial Optimizatio. John Wiley & Sons.

[2] Bäck T., Fogel D., Michalewicz Z. (1997) Handbook of Evolutionary Computation. Oxford University Press.

[3] Bäck T. (1997) Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.

[4] Beasley, J. 1990) - OR Library: distributing test problems by electronic mail. WWW site at http://www.ms.ic.ac.uk/info.html

[5] Bruns, R. (1993) - Direct representation and advanced genetic algorithms for production scheduling. Proceedings of the Fifth International Conference on genetic Algorithms. San Mateo, Morgan Kaufmann.

[6] Davis, L. (1985) - Job Shop Scheduling with Genetic Algorithms. Proceedings of the First International Conference on Genetic Algorithms, pages 136-140. San Mateo; Morgan Kaufmann.

[7] Garey, M. & Johnson, D. (1979) - Computers and Intractability -A guide to the theory of $\mathcal{NP}$-Completeness. Freeman.

[8] Goldberg, D. (1989) - Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

[9] Hart, E. (1997) - The State of the Art in Evolutionary Approaches to Timetabling and Scheduling, Department of Artificial Intelligence, University of Edinburgh. http://www.dcs.napier.ac.uk/evonet.

[10] Michalewicz, Z. (1996) - Genetic Algorithms + Data Structures = Evolution Programs. Third, revised and extended version. Springer-Verlag, USA.

[11] Nakano, R. & Yamada, T. (1991) - Conventional genetic algorithm for Job-Shop Scheduling. Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, Morgan Kaufmann.

[12] Ordóñez, G. & Leguizamón, G. (2001) - . Representaciones Indirectas en Algoritmos Genéticos y la Alcanzabilidad de Schedules Semi-Activos. VII Congreso Argentino de Ciencias de la Computación. Octubre de 2001, El Calafate, Santa Cruz, Argentina. Vol. II Pag. 1229.

[13] Ordóñez, G. & Leguizamón, G. (1999) - Representaciones Indirectas en Algoritmos Geneticos aplicados a un problema Scheduling. V Congreso Argentino de Ciencias de la Computacion. Octubre de 1999, Tandil, Argentina.

[14] Pinedo, Michael.(1995) - Scheduling. Theory, Algorithms and Systems. Columbia University.

[15] Ordóñez, G. & Leguizamón, G. (2004) - Una propuesta de una herramienta flexible para problemas de scheduling. WICC 2004. Mayo de 2004, Neuquen, Argentina.